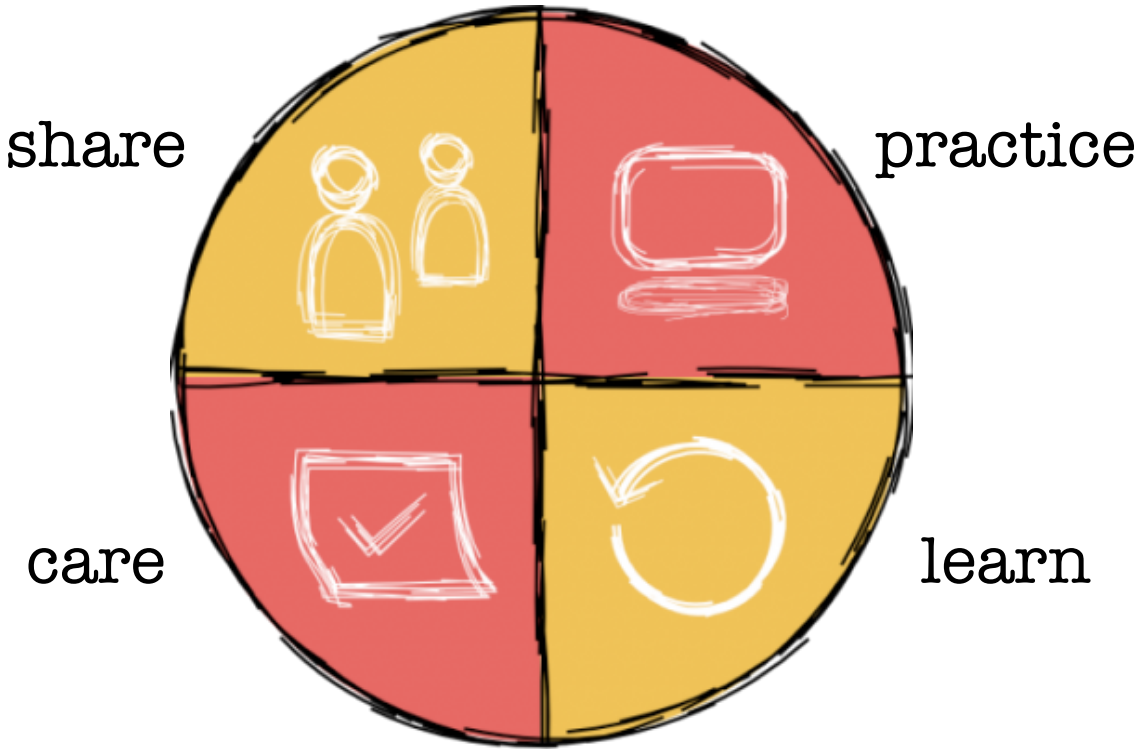
Type Classes for OO programmers

a Scala journey

@ikusalic

Software Craftsmanship



Berlin meetup

Presentation

- Geared toward OO people
- Interrupt with questions/comments/etc.

Agenda

- A bit about types
- Detour - implicits
- Type classes

Types

- Type - a classification identifying type of data
- Determines possible values, operations, and semantic meaning
- E.g. real, integer, bool, Person, Account

Nominal types

- Types are identified by their names
- Names are crucial – e.g. “name conflicts”

Duck typing

- *When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck.* -- James Whitcomb Riley

- Only the part of the structure accessed at runtime is checked for compatibility

Duck typing - example

```
1  "foobar".size
2  >>> 6
3
4  [1, 2, 3].size
5  >>> 3
6
7  42.to_f.size
8  >>> NoMethodError: undefined method
9          'size' for 42.0:Float
```

@ikusalic

Duck typing

- No need to specify types – better abstraction and reuse
- … But no compile-time checking

Structural types

- Types are identified by their structure
- Compile-time checking

Structural types - example

```
1  type WithId = { def getId(): String }
2  def id[T <: WithId](e: T) = e.getId()
3
4  class Foo { def getId() = "foo" }
5  class Bar { def getId() = "bar" }
6
7  id(new Foo)
8  >>> String = foo
9
10 id(new Bar)
11 >>> String = bar
```

@ikusalic

Structural types

- Ad-hoc grouping of types based on structure

- … But what if the method does what we want but is called differently?
- … And we do not control the implementation

Implicits

- Bridge from structural types to type classes
- Custom, scoped implicit conversion rules
- Extending types we do not control

Implicits - usage example

```
1  "foobar " + 42  // Java: "foobar 42"
2
3
4  1.to(4)
5  >>> scala.collection.immutable.Range.Inclusive
6      = Range(1, 2, 3, 4)
```

@ikusalic

Implicits - augmenting

```
1  implicit final class ExtendedInt(val self: Int)
2          extends AnyVal {
3
4      def to(end: Int): Range.Inclusive =
5          Range.inclusive(self, end)
6  }
```

@ikusalic

Implicits - complexity

```
1  def map[B, That](f: A => B)
2      (implicit bf: CanBuildFrom[Repr, B, That]): That
```

Whenever something is even slightly ugly in Scala you introduce an implicit to make it confusing instead -- Peter Fraenkel

@ikusalic

Type classes

- Classify types based on their structure
- Ad-hoc grouping of types

- Extending existing types

Type classes

- Wiki: *a type class is a type system construct that supports ad hoc polymorphism* ?
- “class” ?
- More powerful version of interfaces ?

@ikusalic

Type classes – implementation

- Create an abstract class that will represent my type class
- Create elements representing the classes belonging to my type class
- Use an implicit parameter to restrict the type parameter to my type class

Type classes – ad-hoc grouping

```
1  abstract class Acceptable[T]
2  object Acceptable {
3      implicit object IntOk extends Acceptable[Int]
4      implicit object LongOk extends Acceptable[Long]
5  }
6  def f[T](t: T)(implicit evidence: Acceptable[T]) = t
7
8  f(1)
9  >>> Int = 1
10 f(1L)
11 >>> Long = 1
12 f(1.0)
13 >>> error: could not find implicit value
14             for parameter evidence: Acceptable[Double]
15 // NOTE: example from "Algorithmically challenged" blog
16 // http://dcsobral.blogspot.de/2010/06/
17 //      implicit-tricks-type-class-pattern.html
```

@ikusalic

Type classes – full example

- online example

- Extending existing types
  - even when pure structure does not match

- No need for wrappers on client side

Type classes - recap

- Ad-hoc polymorphism adding constraint to type variable in parametrically polymorph types?

- More powerful version of interfaces that can be attached to types without modifying them?

Type classes – alternatives

- GoF pattern fans will try to use Adapter
  - still need to wrap objects
- Ruby uses monkey patching to augment types whose sources we don't control
  - no compile time safety

Conclusion

- Use Type classes for better abstraction and reuse
- Expand your horizons by learning new languages and new paradigms

Questions ?

Contact me:

@ikusalic

ivan@ikusalic.com

www.ikusalic.com

Thank you

@ikusalic